## Detailed Analysis of the Three Maximal Clique Enumeration Codes

This analysis compares three C++ implementations for enumerating maximal cliques in graphs: `chiba.cpp`, `els.cpp`, and `worst-case.cpp`. Each employs distinct algorithmic strategies, data structures, and optimizations, leading to differences in performance, memory usage, and suitability for specific graph types.

---

### *1. chiba.cpp*

**Algorithm & Approach**:
- **Bron–Kerbosch Variant**: Uses an iterative, stack-based approach inspired by the Bron–Kerbosch algorithm.
- **Vertex Ordering**: Orders vertices by descending degree to prioritize high-degree nodes, similar to Chiba and Nishizeki's work.
- **Maximality Check**: Verifies maximality by ensuring no node in the candidate set `P` can extend the current clique `C`.

**Data Structures**:
- **Adjacency Matrix**: Implemented as a `vector<vector<bool>>`, which is space-efficient for dense graphs but consumes (O(n^2)) memory, making it impractical for large sparse graphs.
- **State Stack**: Uses a stack of `State` objects to simulate recursion, tracking the current clique (`C`), candidate nodes (`P`), and processed nodes.

**Optimizations**:
- **Atomic Counter**: `totalMaximalCliques` uses `atomic<int>`, hinting at potential parallelization (though no threads are explicitly used).
- **Output Batching**: Writes cliques to a file incrementally and logs progress every 1000 cliques.

**Theoretical Considerations**:
- **Time Complexity**: Similar to Bron–Kerbosch $O(m \cdot \alpha(G))$ but optimized by vertex ordering.
- **Space Complexity**: O(n + m) due to the adjacency matrix, limiting scalability.

**Strengths**:
- Simple implementation for small or dense graphs.
- Explicit maximality check ensures correctness.

**Weaknesses**:
- Adjacency matrix is memory-intensive for large graphs.
- No pivot selection or degeneracy ordering, leading to redundant branches in sparse graphs.

**Run Time:**

**wiki-Vote: 575.85sec**

**Email-Enron: 10342.661**

**as-Skitter: Not Recorded**

---

## *2. els.cpp*

**Algorithm & Approach**:
- **ELS Algorithm**: Implements the Bron–Kerbosch variant by Eppstein, Loffler, and Strash (2010), optimized using **degeneracy ordering**.
- **Degeneracy Ordering**: Computes a vertex ordering where each vertex has the smallest possible degree in the remaining graph. This reduces the search space significantly for sparse real-world graphs.
- **Pivot Selection**: Reduces recursion branches by selecting a pivot node with the most connections in the candidate set.

**Data Structures**:
- **Adjacency List**: Uses `vector<vector<int>>`, ideal for sparse graphs ((O(n + m)) space).
- **Hash Sets**: `unordered_set` for tracking candidate (`P`) and excluded (`X`) nodes, though with some overhead.

**Optimizations**:
- **Degeneracy Ordering**: Reduces the worst-case complexity to (O(d n ^{d/3})), where (d) is the graph degeneracy.
- **Milestone Logging**: Prints progress every 10,000 cliques.

**Theoretical Considerations**:
- **Efficiency**: Excels on real-world graphs (e.g., social networks) with low degeneracy.
- **Pivot Heuristic**: Minimizes branching factor by prioritizing high-connectivity pivots.

**Strengths**:
- Optimal for sparse graphs with low degeneracy.
- Memory-efficient adjacency list.

**Weaknesses**:
- Hash set operations (`unordered_set`) may introduce overhead compared to bitmasking.
- No parallelization.

**For Histogram refer webpage**

**Run Time**

**wiki-Vote: 241.54 sec**

**Email-Enron: 233.68**

**as-Skitter: 97776.13**

**For Histogram refer webpage**

---

### *3. worst-case.cpp*

**Algorithm & Approach**:
- **Tomita's Algorithm**: Iterative implementation of Bron–Kerbosch with **Tomita's pivot selection**, designed for robust performance in worst-case scenarios.
- **Pivot Selection**: Chooses a pivot with the maximum degree in (P X) to minimize recursion branches.

**Data Structures**:
- **Adjacency List with Hash Sets**: `vector<unordered_set<int>>` for (O(1)) adjacency checks.
- **Iterative Stack**: Avoids recursion depth limits by using a stack to track states.

**Optimizations**:
- **Fast Input Processing**: Maps vertex IDs to contiguous indices during input parsing.
- **Worst-Case Robustness**: Performs well on adversarial graphs where degeneracy ordering is ineffective.

**Theoretical Considerations**:
- **Time Complexity**: $O(3^{\{n/3\}})$ in the worst case, matching the theoretical lower bound.
- **Space Complexity**: O(n + m), efficient for sparse graphs.

**Strengths**:
- Resilient to graphs with high degeneracy.
- Iterative approach avoids stack overflow.

**Weaknesses**:
- Slower on real-world graphs compared to ELS due to lack of degeneracy optimization.
- `unordered_set` may underperform compared to sorted vectors for cache locality.

**Run Time:**

**wiki-Vote: 4.69sec**

**Email-Enron: 24.98sec**

**as-Skitter: 28869.24sec**

---

## Theoretical Analysis of Each Code

### *1. chiba.cpp*

- **Algorithm**: The algorithm is a variant of the Bron–Kerbosch method, which is a classic backtracking algorithm for enumerating maximal cliques. The key feature here is the use of a stack to simulate recursion, making it iterative rather than recursive. This avoids potential stack overflow issues in deep recursion trees.
- **Vertex Ordering**: Vertices are ordered by degree in descending order. This heuristic aims to prioritize high-degree vertices, which are more likely to be part of larger cliques, potentially reducing the search space.
- **Maximality Check**: The algorithm explicitly checks for maximality by ensuring that no node in the candidate set P can extend the current clique C. This is a standard requirement in maximal clique enumeration to avoid redundant cliques.
- **Time Complexity**: The worst-case time complexity is $\mathbf{O(3^{\{n/3\}})}$, which is the theoretical lower bound for maximal clique enumeration. This is because the algorithm explores all possible subsets of vertices that form cliques.
- **Space Complexity**: The use of an adjacency matrix results in $\mathbf{O(n^2)}$ space complexity, which is inefficient for large sparse graphs. The stack-based approach adds (O(n)) space for the recursion simulation.
- **Optimizations**: The use of an atomic counter suggests potential for parallelization, though it is not utilized in the current implementation. The algorithm also logs progress every 1000 cliques, which is useful for monitoring long-running computations.

### *2. els.cpp*

- **Algorithm**: This implementation uses the ELS algorithm, which is a variant of Bron–Kerbosch optimized for sparse graphs. The key innovation is the use of **degeneracy ordering**, which significantly reduces the search space by processing vertices in a specific order.
  - **Degeneracy Ordering**: The algorithm computes a degeneracy ordering, where vertices are processed in order of their degree in the remaining graph. This ordering ensures that the algorithm focuses on the most constrained parts of the graph first, reducing the number of recursive calls.
- **Pivot Selection**: The algorithm uses a pivot selection heuristic to minimize the branching factor. The pivot is chosen as the vertex with the most connections in the candidate set P, which helps in reducing the number of recursive branches.
- **Time Complexity**: The time complexity is $\mathbf{O(d.n.3^{(d/3)})}$, where (d) is the degeneracy of the graph. This is a significant improvement over the general Bron–Kerbosch algorithm for sparse graphs, where (d) is typically much smaller than (n).
- **Space Complexity**: The adjacency list representation results in $\mathbf{(O(n + m))}$ space complexity, which is efficient for sparse graphs. The use of hash sets for P and X adds some overhead but is generally manageable.

- **Optimizations**: The algorithm logs progress every 10,000 cliques, which is useful for tracking long-running computations. The degeneracy ordering and pivot selection are key optimizations that make this algorithm particularly effective for real-world graphs.

### *3. worst-case.cpp*

- **Algorithm**: This implementation uses Tomita's algorithm, which is another variant of Bron–Kerbosch. The key feature is the use of **Tomita's pivot selection**, which is designed to handle worst-case scenarios effectively.
- **Pivot Selection**: The pivot is chosen as the vertex with the maximum degree in (P X). This heuristic aims to minimize the number of recursive branches by focusing on the most connected vertices.
- **Iterative Approach**: The algorithm uses an iterative stack-based approach to avoid recursion depth limits. This is particularly useful for large graphs where recursion could lead to stack overflow.
- **Time Complexity**: The worst-case time complexity is **$O(3^{(n/3)})$**, matching the theoretical lower bound for maximal clique enumeration. This makes the algorithm robust in adversarial scenarios.
- **Space Complexity**: The adjacency list with hash sets results in **$O(n + m)$** space complexity, which is efficient for sparse graphs. The iterative stack adds ($O(n)$) space for recursion simulation.
- **Optimizations**: The algorithm includes fast input processing, which maps vertex IDs to contiguous indices, improving efficiency. The use of Tomita's pivot selection ensures robustness in worst-case scenarios, though it may be slower on real-world graphs compared to ELS.

## Comparison of the Three Implementations

| Feature | Chiba Implementation | ELS Implementation | Worst-Case Implementation |
|---|---|---|---|
| **Algorithm** | Bron-Kerbosch | Tomita (Bron-Kerbosch + Pivot) | Tomita (Bron-Kerbosch + Pivot) |
| **Pivot Selection** | No | Yes | Yes |
| **Graph Representation** | Adjacency List | Adjacency List | Adjacency List |
| **Time Complexity** | $O(m \cdot \alpha(G))$ | $O(d \cdot n \cdot 3^{d/3})$ | $O(3^{(n/3)})$ |

| | | | |
|---|---|---|---|
| **Space Complexity** | $O(n^2)$ | $O(n + m)$ | $O(n + m)$ |
| **Optimization** | Degree-based ordering | Degree-based ordering | Pivot Selection |
| **Output** | Cliques, Histogram, Time | Cliques, Histogram, Time | Cliques, Histogram, Time |

## *Theoretical Analysis*

1. **Degeneracy vs. Pivot Heuristics**:
   - ELS leverages degeneracy ordering to exploit sparsity in real-world graphs, reducing practical runtime.

   - Tomita's pivot selection ensures robustness but lacks domain-specific optimizations.
2. **Adjacency Representation**:
   - Matrix-based approaches (chiba.cpp) are limited by memory but allow $O(1)$ edge checks.

   - List-based approaches (els.cpp, worst-case.cpp) scale better but require $O(k)$ edge lookups.
3. **Maximality Verification**:
   - `chiba.cpp` explicitly checks maximality, while others rely on algorithmic correctness.